UNIVERSITÉ DE MONTRÉAL

CENTRE DE RECHERCHE SUR LES TRANSPORTS

CASE POSTALE 6128, SUCCURSALE A, MONTRÉAL, QUÉBEC H3C 3J7

AN ALGORITHM FOR THE SHORTEST PATH PROBLEM WITH RESOURCE CONSTRAINTS

by

Martin Desrochers

Cette étude a été publiée grâce à une subvention du fonds F.C.A.R. pour l'aide et le soutien à la recherche.

Université de Montréal
Centre de recherche sur les transports - Publication #421A
June 1986

Dépôt légal - 3e trimestre 1986

SUMMARY

We present a primal-dual algorithm which uses dynamic programming to calculate the shortest path in a network G=(N,A) while respecting time window constraints of the form $[a_i^\ell, b_i^\ell]$ $\ell=1,\ldots,L$. A primal function and a dual function are defined at each node based on the known paths. The algorithm calculates new feasible primal and dual functions. The algorithm runs in pseudo-polynomial time. We present the results of numerical tests for problems with up to 1000 nodes and L=5 additional constraints. The most surprising conclusion of these tests is that as the number of constraints increases, the execution time does not increase as the number of dynamic programming states increases, but rather decreases because the number of feasible states examined decreases.

RÉSUMÉ

Nous présentons un algorithme primal-dual utilisant la programmation dynamique pour calculer dans un réseau G=(N,A) le plus court chemin respectant de multiples contraintes de type fenêtres de temps, $[a_i^\ell, b_i^\ell]$ $\ell=1,\ldots,L$. Une fonction primale et une fonction duale sont définies à chaque noeud à partir des chemins connus. L'algorithme calcule de nouveaux chemins primaux ou duaux réalisables de façon à réduire l'écart entre les fonctions primale et duale. L'algorithme termine en temps pseudo- polynomial. Nous présentons les résultats de tests numériques pour des problèmes ayant jusqu'à 1000 noeuds et L=5 contraintes supplémentaires. La conclusion la plus surprenante des tests numériques est que, lors de l'augmentation du nombre de contraintes, le temps d'exécution de l'algorithme n'augmente pas quand le nombre d'états de la programmation dynamique augmente mais plutôt diminue car le nombre d'états réalisables examinés diminue.

One of the most common subproblems encountered during the solution of routing, scheduling and telecommunications problems is the shortest path problem. In some models, additional constraints are imposed on the subproblem. The algorithm described in this paper was developed for the solution of subproblems encountered during the construction of work schedules for urban bus drivers using a column generation method.

1. THE PROBLEM

Let G=(N,A) be a network where A is the set of arcs and N is the set of nodes including a source p and a sink q. A positive or negative cost $c_{ij} \in R$ is associated with each arc $(i,j) \in A$, and the shortest path problem is defined as follows where x_{ij} represent the flow on arc (i,j) :

$$\text{Minimize} \quad c.x = \sum_{(i,j)\in A} c_{ij}\, x_{ij} \tag{1}$$

subject to

$$x \in X(G) = \begin{cases} \sum_{j\in N} x_{ij} - \sum_{j\in N} x_{ji} = \begin{cases} +1, & i=p \\ 0, & i\in N,\ i\neq p,\ i\neq q \\ -1, & i=q \end{cases} & (2) \\ x \geq 0 & (3) \end{cases}$$

The problem also includes constraints on the use of L resources. With each arc (i,j) and with each resource ℓ is associated $d^{\ell}_{ij} \in N$, the quantity of resource ℓ used on arc (i,j). When the nodes represent tasks, the cost and resources used by node i are added to those on the outgoing arcs (i,j). Let T^{ℓ}_1 be the quantity of resource ℓ used by following a path from the source to node j; the problem consists of determining the elementary path of minimum cost between p and q such that T^{ℓ}_j respects the feasibility window $[a^{\ell}_j, b^{\ell}_j]$ for all nodes of the path and for all resources $\ell=1,\ldots,L$. We consider the problem where it is possible to reach a node using less of a resource ℓ than the minimum of the feasible window for this resource. By wasting the resource ℓ, it is thus possible to begin visiting

this node at the minimum of the feasible window for this resource. The shortest path problem with time windows [DESR83] is one example of this; it is possible to arrive at a node before the beginning of the time window and to wait (i.e. waste time) before visiting the node at the beginning of the time window.

To obtain the shortest path problem with resource constraints (SPPRC), we must add the following constraints to the shortest path problem (1-3) with T_i^ℓ representing the quantity of the ℓth resource used to arrive at node i :

$$x_{ij} > 0 \Rightarrow T_i^\ell + d_{ij}^\ell \leq T_j^\ell \qquad (i,j,)\in A,\ \ell=1,..,L \qquad (4)$$

$$a_i^\ell \leq T_i^\ell \leq b_i^\ell \qquad i\in N,\ \ell=1,...,L \qquad (5)$$

If resource wastage is not permitted, the inequality in equation (4) must be replaced by an equality. The resulting problem is more constrained and the optimal solution may have a higher cost. This model corresponds to the classical dynamic programming model (see Jaffe [JAFF84]) and is generally more difficult to solve than when wastage is permitted.

All infeasible arcs are considered to be excluded from the set A. An arc (i,j) is infeasible if it is impossible to go from node i to node j while respecting the feasibility windows for i and j. All arcs $(i,j)\in A$ thus respect the following condition for all resources :

$$a_i^\ell + d_{ij}^\ell \leq b_j^\ell \qquad \ell=1,...,L \qquad (6)$$

Well known problems are special cases of the shortest path problem with resource constraints : the multidimensional knapsack and resource constrained shortest paths. Showing that the multidimensional knapsack (known to be NP-HARD) is a special case of the SPPRC proves that SPPRC is NP-HARD [GARE 79]. The multidimensional knapsack is defined as

$$\text{Max} \sum_{j=1}^{M} c_j x_j$$

$$\text{subject to} \sum_{j=1}^{M} d_j^{\ell} x_j \leq b^{\ell} \qquad \ell=1,\ldots,L$$

$$x_j \in \{0,1\}.$$

To obtain an equivalent SPPRC, we define a network $G=(N,A)$ with arcs

1) (p,i) from the source to an object i, $i=1,\ldots,M$

2) (i,q) from any object i, $i=1,\ldots,M$, to the sink

3) (i,j) if $i < j$ and $d_i^{\ell} + d_j^{\ell} \leq b^{\ell}$, $\ell=1,\ldots,L$. The first condition prevents multiple uses of an object and the second prevents obvious infeasibilities.

The cost of the SPPRC are defined as $c_{ij} = -c_j$, the resources consumption as $d_{ij}^{\ell} = d_j^{\ell}$ and the windows $[a_i^{\ell},b_i^{\ell}] = [0,b^{\ell}]$. Any feasible path from the source to the sink corresponds to a feasible multidimensional knapsack and the minimum cost path corresponds to the maximum cost multidimensional knapsack. Thus the multidimensional knapsack is a special case of the SPPRC and the SPPRC is NP-HARD.

The resource constrained shortest path (RCSP) [ANEJ 84, BEAS 84] is also a special case of the SPPRC. The RCSP is formulated using constraints (1-3) and the following constrains :

$$a^{\ell} \leq d_{ij}^{\ell} x_{ij} \leq b^{\ell} \qquad \ell=1,\ldots,L.$$

To obtain an equivalent SPPRC, we only need to define the windows as $[a_i^{\ell},b_i^{\ell}] = [-\infty,\infty]$ for all nodes except the sink and as $[a_q^{\ell},b_q^{\ell}] = [a^{\ell},b^{\ell}]$ for the sink. Aneja et al. [ANEJ 84] indicate how to compute tighter windows on intermediate nodes $N-\{p,q\}$ using shortest paths problems.

The multicriteria shortest path problem is formulated using constraints (2-3) and the objective function :

$$\text{Min} \begin{cases} \sum_{(i,j)\in A} c_{ij} x_{ij} \\ \sum_{(i,j)\in A} d^{\ell}_{ij} x_{ij} \qquad \ell = 1, \ldots, L. \end{cases}$$

This problem is a SPPRC with infinite windows $[-\infty, \infty]$. The algorithm developed in this paper can solve this problem but the pseudo-polynomial worst case complexity does not hold because there can be an exponential number of efficient paths [WARB 86].

The construction of work schedules for bus drivers involves the determination of a set of workdays which enable all the nodes to be visited while minimizing the total cost. We use a column generation method, similar to that proposed by Desrosiers, Soumis and Desrochers [DESR 84] to solve the problem of school bus schedule construction.

The method consists of constructing feasible workdays until a solution is found which optimally covers the entire set of nodes. At each iteration of the method, we seek to improve the present solution by entering into the basis the workday of minimal marginal cost obtained by solving a constrained shortest path problem.

The difference between the two applications lies in the subproblem solved. In the case of school bus scheduling, a shortest path problem with one additional constraint is solved [DESR83, DESR85a, DESR85b], while in the case of the construction of workdays, the subproblem must deal with several resource constraints. These resource constraints include restrictions on the total length of the workday, the number of nodes in a day, its useful length, etc.

2. LITERATURE REVIEW

Although the shortest path problem with resource constraints and the related multi-criteria shortest path problem have been the focus of 15 or so articles, we will discuss here only those which use our approach :

dynamic programming. For a more complete review of the literature, see Desrochers and Soumis [DESR85a].

We first introduce a definition of the states which is common to most dynamic programming approaches to the problem. With each path X_{pj} from the origin to node j, is associated a label (quantity of resources used and the cost of the path X_{pj}). These labels are represented by (T_j^{1k}, T_j^{2k}, ..., T_j^{Lk}, C_j^k) denoting the characteristics of the kth path X_{pj}^k, where the indices k may be absent if the context is unambiguous. In addition, $T_j^{\cdot k}$ or $T_j^{\cdot}$ will be used to represent the vector of dimension L. For the treatment of these problems by dynamic programming,, the set of states is $\{T_j^{\cdot k} |$ for all k and j$\}$ and C_j^k is the cost associated with the state $T_j^{\cdot k}$. A path X_{pj}^k and its associated state $T_j^{\cdot k}$ are feasible if $a_j^\ell \leq T_j^{\ell k} \leq b_j^\ell$ for all resources $\ell = 1,\ldots,L$. In the case of multi-criteria shortest path problems, the definition of states is the same, but the algorithm optimizes on all components of the state rather than simply the cost.

Calculation of successors (reaching)

There are two methods of this type. The first method, which is based on a generalization of the Ford-Bellman algorithm [JOKS66, VINC74, HABE81, DESR83, JAFF84], works by correcting the labels. The second method which is based on a generalization of Dijkstra's algorithm [MINO75, HANS80, ANEJ83, MART84, DESR85A, WARB86], uses permanent marking of the labels. The basic step in each of the two algorithms is the "reaching" step. Starting with a label associated with a path X_{pi}, the method determines the labels corresponding to feasible paths X_{pj} which extend the path X_{pi} by adding all arcs $(i,j) \in A$. For each node i, this algorithm creates new labels for several nodes j; in other words, several labels are created for node j if this node has several predecessors. The labels at each node j must therefore be updated several times. This is costly especially if there are several additional constraints.

Search of predecessors (pulling)

This primal-dual method [DESR85b], developed for the reoptimization of the shortest path problem with time windows, calculates the paths X_{pj} associated with some or all of the states of a node j. For a set S of states at node j, the "pulling" process consists of creating labels ($T_j^* = (\max \{ a_j^\ell, T_j^{\ell k} + d_{ij}^\ell \}, \ell=1,\ldots,L), C_i^k + c_{ij}$) associated with paths X_{pj} obtained by extending all paths X_{pi} for which the addition of arc $(i,j) \in A$ allows arrival at node j in a state $T_j^* \in S$. As the new labels are created at a single node, this algorithm requires the updating of labels at only one node, in contrast with the reaching approach.

3. THE PRINCIPLES OF MULTIDIMENSIONAL PULLING

The principles of pulling were developed by Desrochers and Soumis [DESR85b] for the case with one additional constraint. It is possible to generalize this for the shortest path problem with several additional constraints.

Definition 1

Let X^1 and X^2 be two distinct paths from p to j with associated labels (T_j^{*1}, C_j^1) and (T_j^{*2}, C_j^2). Then X^1 dominates X^2 or $(T_j^{*1}, C_j^1) < (T_j^{*2}, C_j^2)$ if and only if $C_j^1 < C_j^2$, $T_j^{\ell 1} < T_j^{\ell 2}$ for all $\ell=1,\ldots,L$ et $(T_j^{*1}, C_j^1) \neq (T_j^{*2}, C_j^2)$.

The dominance relation allows us to conclude that a path X_{pj} and its associated label (T_j^*, C_j) are non-dominated (efficient) if and only if X_{pj} is the least cost path arriving at node j with values less than or equal to T_j^ℓ $(\ell=1,\ldots,L)$.

We use Bellman's optimality principle (i.e. that all subpaths of an efficient path must also be themselves efficient) to define the following recurrence equations which characterize the efficient paths from the source to all the nodes for the SPPRC :

For all $j \in N$ and all feasible T_j^* :

$$C(j,T_j^*) = \min_{\substack{i \in N,\ T_j^* \text{ feasible state}}} \{C(i,T_i^*)+c_{ij} \mid T_i^\ell + d_{ij}^\ell \leq T_j^\ell, (i,j) \in A. \quad \ell=1,\ldots,L\} \qquad (8)$$

$$C(p,0^*) = 0 \qquad (9)$$

The application of the optimality principle accelerates the calculation of the recurrence equations (8-9) by avoiding the extension of dominated paths X_{pi}^k and thus the creation of new dominated paths.

As the dominance relation is not a total ordering, it is not possible to order all the paths. This means that there may be several efficient paths for each node. We denote by $EFF\{R_j\}$, the undominated labels of the set of labels R_j. Figure 1 shows for one node, the relationship between different (value, cost) labels associated with efficient paths (X^1 to X^4), and dominated paths (X^5 to X^8) for the case with a single additional constraint.

Using the dominance relation, we can define the cost of the minimum cost path as a function of the state. The entire set of labels $R_j = \{(T_j^{*k}, C_j^k)\}$ associated with node j is used to define the function. Figure 1 illustrates the function for a simple example.

Let R_j be the set of labels associated with the paths X_{pj}^k. Then, for all the states T_j^* at node j, the function $F_R(T_j^*)$ is defined by :

$$F_R(T_j^*) = \min \{C_j^k \mid T_j^{\ell k} \leq T_j^\ell, \quad \ell=1,\ldots,L, \text{ for } (T_j^{*k}, C_j^k) \in R_j\}.$$

The function $F_R(T_j^*)$ represents the minimum cost of a path X_{pj} which uses less than T_j^* units of resources.

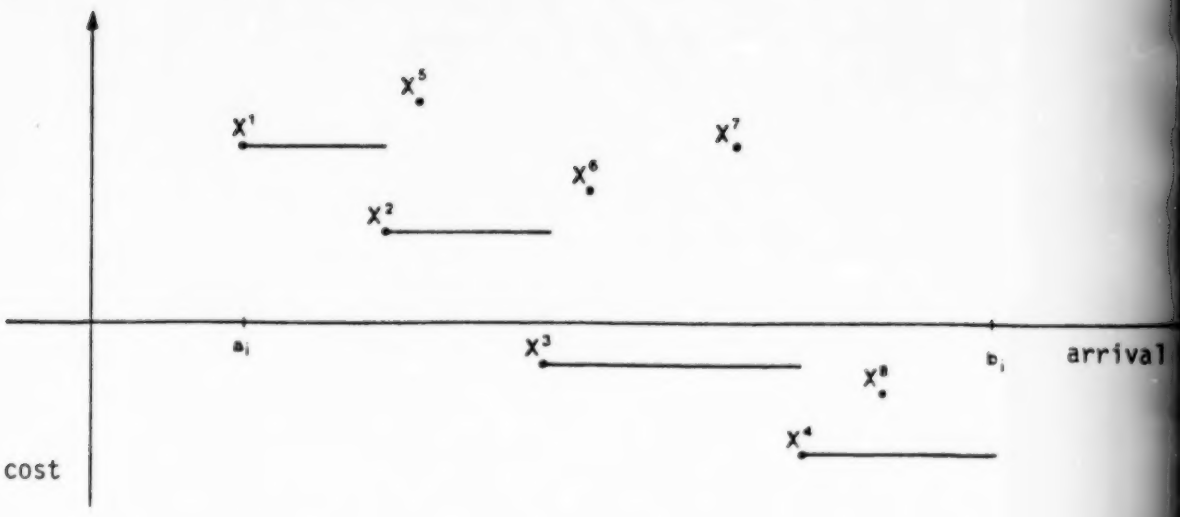

Figure 1 - Dominance relation between labels associated with different paths.

The algorithm requires the creation of two sets of labels at each node. Each set of labels defines a function. The algorithm modifies the sets of labels, thus modifying the associated functions, until the solution is obtained. The first set P_j includes the labels associated with feasible paths and defines the primal function. The primal function provides an upper bound on the cost of efficient solutions at each node. If P_j is empty, the value of the primal function is defined as being infinitely large. The second set Q_j includes the labels associated with paths which may or may not be feasible. The labels of Q must not be dominated by labels associated with feasible paths. The labels of Q_j define the dual function which gives a lower bound for the cost of efficient solutions at each node.

P_j and Q_j can be initialized in two ways. If we have no prior informations, P_j is set at $\emptyset$ and the label $(a_j^1,\ldots,a_j^\ell,-\infty)$ is used to initialize Q_j. In the case of a reoptimization after an increase in costs,

Q_j is initialized with all the old labels as these still represent lower bounds, while P_j will contain only the labels associated with paths which are still feasible as these paths will remain optimal.

When the primal and dual functions have the same value, the label in P_j and Q_j for this state is associated with an efficient path for this state. If the primal and dual functions at a state have different values, there is uncertainty about the cost of the efficient path for this state. A zone of uncertainty is a state or set of adjacent states for which the primal and dual functions have different values. A zone of uncertainty is identified by reference to the label in $Q_j - P_j$ which dominates the other labels associated with the zone of uncertainty.

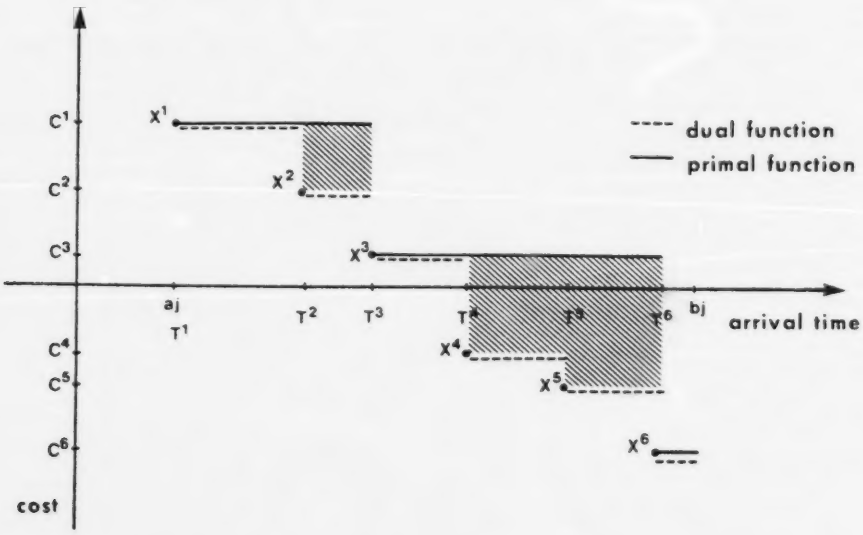

Figure 2 - Examples of primal and dual functions.

Figure 2 illustrates the primal and dual functions for the set $P_j = \{(X^1,C^1),(X^3,C^3),(X^6,C^6)\}$ and the set $Q_j = \{(X^1,C^1),(X^2,C^2),(X^3,C^3),(X^4,C^4),(X^5,C^5),X^6,C^6)\}$. The states in the shaded area form the zone of uncertainty.

There are two ways of reducing the size of the zone of uncertainty. One may either increase the value of the dual function, or reduce the value of the primal function. The numerical example in section 6 illustrates this process in detail.

The first way of reducing uncertainty involves the calculation of a new dual function for this area. This new lower bound is calculated by finding all paths X_{pi} whose extension by an arc $(i,j)\varepsilon A$ produces a path X_{pj} with the state corresponding to its label in the bounded zone. The subset R_j of efficient labels is calculated in order to obtain the labels defining the lower bound :

$$R_j = EFF\{(T_j^*, C_j) = ((\max\{a_j^\ell, T_i^{\ell k} + d_{ij}^\ell\},\ \ell=1,\ldots,L),\ C_i^k + c_{ij}) \mid$$
for all $(i,j)\varepsilon A$, $(T_i^{*k}, C_i^k)\varepsilon\ Q_i$ and T_j^* belonging to the zone to be explored$\}$.

The other way of reducing uncertainty is to improve the primal function by the calculation of feasible solutions in the bounded zone. There is presently no feasible solution in the zone of uncertainty. Feasible solutions are determined by finding all the feasible paths X_{pi} whose extension by an arc $(i,j)\varepsilon A$ produces a path X_{pj} with the state corresponding to its label lying in the bounded zone. The set RP_j of feasible solutions making up the new primal function in the bounded zone is calculated as follows :

$$RP_j = EFF\{(T_j^*, C_j) = ((\max\{a_j^\ell, T_i^{\ell k} + d_{ij}^\ell\},\ \ell=1,\ldots,L),\ C_i^k + c_{ij}) \mid$$
for all $(i,j)\varepsilon A$, $(T_i^{*k}, C_i^k)\varepsilon P_i$ and T_j^* belonging to the zone to be explored$\}$.

Some labels in RP_j may be dominated by infeasible labels in R_j.

The replacement of eliminated labels is carried out :

1) by finding the eventual new feasible labels (εRP_j) and efficient labels (εR_j) to be added to P_j. This addition of labels modifies

both the sets defining the upper bound and the lower bound, and destroys the zone of uncertainty in the point :

$$P_j \leftarrow P_j \cup (R_j \cap RP_j)$$

This replacement rule guarantees that P_j will only include efficient solutions.

2) by adding efficient labels (εR_j) to the set Q_j after having eliminated the replaced label :

$$Q_j \leftarrow \{Q_j - (T_j^k, C_j^k)\} \cup R_j$$

to modify the lower bound.

4. ORDER OF REPLACEMENT OF THE LABELS

When replacing an eliminated label, we need to ensure that this label will not reappear producing a step without reduction in the size of the zone of uncertainty. In [DESR85a] an order of treatment of the labels and some qualifying conditions on the arcs are defined to solve the SPPRC for the special case with one constraint using a primal algorithm. The same concepts are used here to define an order of replacement of the labels in $S = \sum_{j \in N} (Q_j - P_j)$ defining a primal-dual algorithm for the SPPRC.

Definition 2.

Let $a, b \in |R^L$. Then a is said to be lexicographically smaller than b ($a \overset{L}{<} b$) if and only if $a \neq b$, and the first non-null element of b-a is positive.

Definition 3.

An arc (i,j) is lexicographically positive if and only if $(0,0\ldots0) \overset{L}{<} (d_{ij}, c_{ij})$.

The lexicographic ordering is a total ordering in $|R^L$ which is compatible with the partial dominance ordering.

The concept of a generalized bucket

A bucket B(S) is a set of labels in S which can be replaced with reduction of the zone of uncertainty. Two properties are important in the formal definition of a bucket :

1) all labels in B(S) must be ready to be replaced.
2) information is available on whether it is possible to delay the replacement of a label in B(S).

This last piece of information allows us, in the case of a large network where only part of the data on the network are available, to delay as long as possible the treatment of unavailable areas of the network.

The concept of a generalized bucket defined by Desrosiers and Soumis [DESR85a] provides this information.

Let $(d^{\bullet}_{\cdot j}, c_{\cdot j}) = \underset{(i,j)\in A}{\text{minlex}} \{(d^{\bullet}_{ij}, c_{ij})\}$, the width of the bucket.

Let $F(S) = \underset{(T^{\bullet}_i, C^k_i)\in S}{\text{minlex}} \{(T^{\bullet k}_i, C^k_i)\}$, the label to be replaced.

We can define the bucket B(S), a set of labels to be replaced :

$$B(S) = \{T^{\bullet k}_i, C^k_i) \in S \mid F(S) \overset{L}{<} (T^{\bullet k}_i, C^k_i) \overset{L}{<} F(S) + (d^{\bullet}_{\cdot i}, c_{\cdot i})\}$$

Theorem 1.

If $(d^{\bullet}_{\cdot i}, c_{\cdot i}) \overset{L}{>} (0, \ldots, 0)$ for all $i \in N$, then on replacing a label in B(S),

1) it is impossible to reproduce the label being replaced;
2) the zone of uncertainty is reduced.

Proof.

1) Suppose it were possible to reproduce the label $(T^{\bullet}_j, C_j) \in B(S)$. Then there exists a label $(T^{\bullet}_i, C_i) \in S$ such that

$$(T^{\bullet}_i, C_i) + (d^{\bullet}_{ij}, c_{ij}) = (T^{\bullet}_j, C_j) \overset{L}{<} F(S) + (d^{\bullet}_{\cdot j}, c_{\cdot j})$$

$$\Rightarrow (T_i^{\bullet}, C_i) + (d_{ij}^{\bullet}, c_{ij}) \overset{L}{<} F(S) + (d_{.j}^{\bullet}, c_{.j})$$

$$\Rightarrow (T_i^{\bullet}, C_i) \overset{L}{<} F(S) \quad \text{car} \quad (d_{ij}^{\bullet}, c_{ij}) \overset{L}{>} (d_{.j}^{\bullet}, c_{.j}) \quad \text{by definition.}$$

This contradicts the definition of F(S).

2) The algorithm calculates a new dual function by computing the new feasible and infeasible elements of the lower bound for the zone of uncertainty as follows :

$$R_j = EFF\{(T_j^{\bullet}, C_j) = ((\max\{a_j^{\ell}, T_i^{\ell k} + d_{ij}^{\ell}\}, \ell = 1, \ldots, L), C_i^k + c_{ij}) \mid (T_i^{\bullet k}, C_i^k) \in Q_i, \ (i,j) \in A$$

and $T_j^{\bullet}$ belonging to the zone of uncertainty$\}$.

As the feasible labels of R_j eliminate uncertainty by defining the dual and the primal functions simultaneously, only the infeasible labels of R_j, i.e. $\notin RP_j$ define the uncertainty due to the gap between the two functions. A lexicographic lower bound on the value of the smallest infeasible label in R_j is :

$$((\max\{a_j^{\ell}, T_j^{\ell} + d_{.j}^{\ell}\}, \ell = 1, \ldots, L), C_j + c_{.j}).$$

In fact, $((\max\{a_j^{\ell}, T_j + d_{.j}^{\ell}\}, \ell = 1, \ldots, L), C_j + c_{.j}) \overset{L}{<}$

$$((\max\{a_i^{\ell}, T_i^{\ell k} + d_{ij}^{\ell}\} \ell = 1, \ldots, L), \ C_i^k + c_{ij})$$

for all $(T_i^{\bullet k}, C_i^k) \in D$ et $(i,j) \in A$.

This lower bound on the first infeasible path introduced during the replacement of $(T_j^{\bullet k}, C_j^k) \in B(S)$ guarantees that for all states T such that

$$T_j^{\bullet} \overset{L}{<} T^{\bullet} \overset{L}{<} (\max\{a_j^{\ell}, T_j^{\ell} + d_{.j}^{\ell}\}, \ell = 1, \ldots, L)$$

the new labels calculated will belong to both R_j et RP_j, i.e. for all these states, the primal function has the same value as the dual function, thus reducing the size of the original zone of uncertainty.

Q.E.D.

5. THE ALGORITHM

Theorem 1 can be used to develop the following algorithm for the solution of the SPPRC.

Step 1. Initialization

- preparation of the sets P_j et Q_j.
- calculation of $(d^{\bullet}_{\bullet j}, c_{\bullet j})$ for each node.

Step 2. Calculation of the bounds of the bucket

Find the label $(T_j^{\bullet k}, C_j^k)$ of minimum lexicographic cost from the set $S = \underset{j \in N}{U} (Q_j - P_j)$. If $S = \emptyset$, stop.

Step 3. Search for a label $(T_j^{\bullet k}, C_j^k)$ to be treated

- Find a label belonging to the first bucket.
- If the bucket is empty, return to step 2.

Step 4. Choice of the zone to be explored within the zone of uncertainty

Step 5. Replacement of the label $(T_j^{\bullet k}, C_j^k)$

A) Calculation of the new dual function in the zone to be explored :

$$R_j = EFF\{(T_j^{\bullet}, C_j) = ((\max\{a_j^{\ell}, T_i^{\ell k} + d_{ij}^{\ell}\}, \ell = 1, \ldots, L), C_i^k + c_{ij}) \mid$$
$$\text{for all } (i,j) \in A,\ (T_i^{\bullet k}, C_i^k) \in Q_i$$
$$\text{and } T_j^{\bullet} \text{ in the zone to be explored.}\}$$

B) Calculation of the primal function in the zone to be explored :

$$RP_j = EFF\{(T_j^{\bullet}, C_j) = ((\max\{a_j^{\ell}, T_i^{\ell k} + d_{ij}^{\ell}\}, \ell = 1, \ldots, L), C_i^k + c_{ij}) \mid$$
$$\text{for all } (i,j) \in A,\ (T_i^{\bullet k}, C_i^k) \in P_i$$
$$\text{and } T_j^{\bullet} \text{ in the zone to be explored.}\}$$

C) Creation of the new sets Q_j et P_j

- $P_j = P_j \cup (R_j \cap RP_j)$
- $Q_j = (Q_j - \{(T_j^k, C_j^k)\}) \cup R_j$

return to step 3.

Only step 4 (the choice of the zone to be explored within the zone of uncertainty) remains to be specified. In the numerical example, the entire zone of uncertainty will be explored.

6. NUMERICAL EXAMPLE

The following numerical example illustrates the solution of a shortest path problem with one additional constraint. This is a problem with 4 nodes, 5 arcs and 22 states as shown in Figure 3. The source (node P) is initialized according to (9). The sets P_j are empty for the other nodes. The set Q_j is initialized according to (9) for the source, and with the label $(a_j, -\infty)$ for the other nodes. Using these initializations, the primal and dual functions can be calculated, and the zones of uncertainty can be determined as shown in Figure 3.

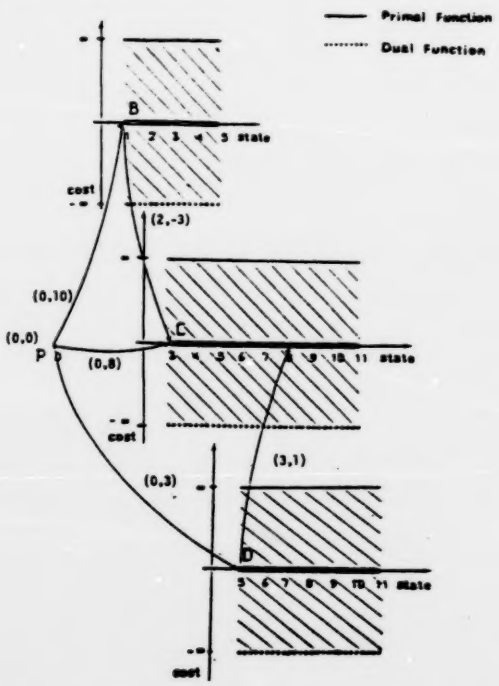

Figure 3 - Initialization of the network.

To simplify the explanation of the algorithm, the example given has only one constraint. To simplify the example further, we combine steps 2 and 3 by always choosing the smallest element of the bucket B(S), eliminating the need to recalculate the limits of the bucket at each iteration.

At the first iteration, the set S of labels to be replaced includes the labels $(1,-\infty)$ at node B, $(3,-\infty)$ at node C and $(5,-\infty)$ at node D. We choose to treat the label $(1,-\infty)$ at node B. The source is the only predecessor and we obtain the label (1,10) associated with a feasible path, the sets $Q_B = P_B = \{(1,10)\}$, and the elimination of the uncertainty at this node. The result is illustrated in Figure 4.

At the beginning of the second iteration, the set S includes $\{(3,-\infty), (5,-\infty)\}$. We choose to search for predecessors of the label $(3,-\infty)$ at node C. The zone of uncertainty includes all the states of the node C. This node has three predecessors P, B and D. We obtain the labels $\{(3,8), (3,7), (8,-\infty)\}$ and the sets $Q_C = \{(3,7), (8,-\infty)\}$ and $P_C = \{(3,7)\}$. Uncertainty has been eliminated at the states associated with the values 3 to 7. The result is illustrated in Figure 5.

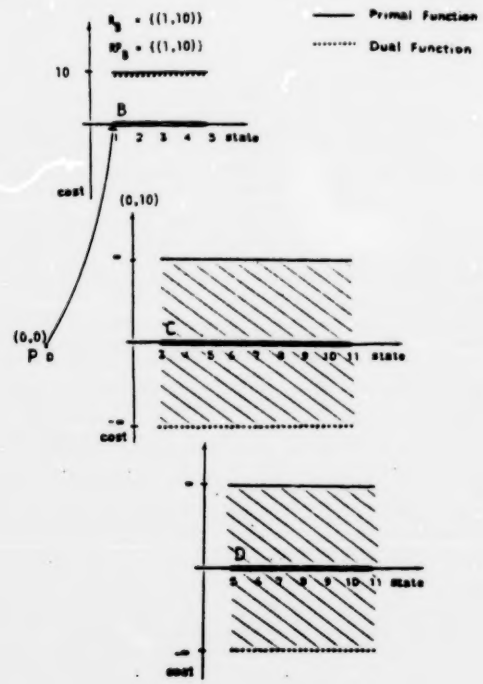

Figure 4
Pulling of label $(1,-\infty)$ at node B

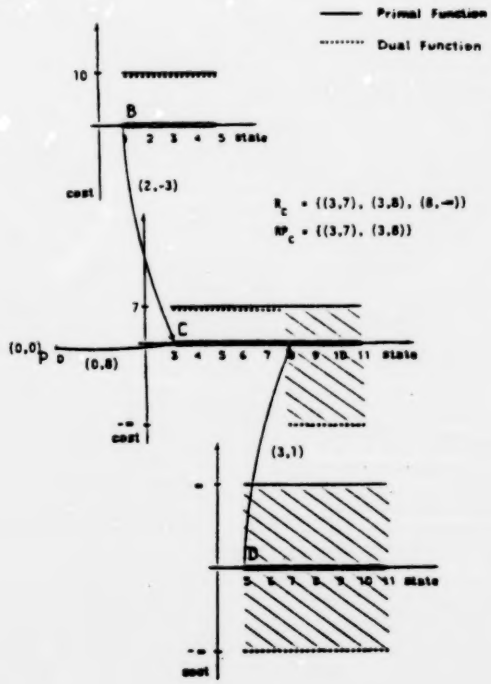

Figure 5
Pulling of label $(3,-\infty)$ at node C

At the beginning of the third iteration, the set S includes $\{(5,-\infty), (8,-\infty)\}$. We choose to treat the label $(5,-\infty)$ at node D. The zone of uncertainty includes all the states at node D. This node has only one predecessor : the source. We obtain a label (5,3) and the sets $P_D=Q_D=\{(5,3)\}$. The uncertainty at this node is eliminated. The result is illustrated in Figure 6.

At fourth iteration, the set S includes only the label $(8,-\infty)$ at node C. The zone of uncertainty includes four states associated with the values 8 to 11. This node has three predecessors, so we obtain the labels $\{(8,8), (8,7), (8,4)\}$. $P_C=Q_C = \{(3,7), (8,4)\}$, and the uncertainty is eliminated at this node and for the problem. The final solution is illustrated in Figure 7.

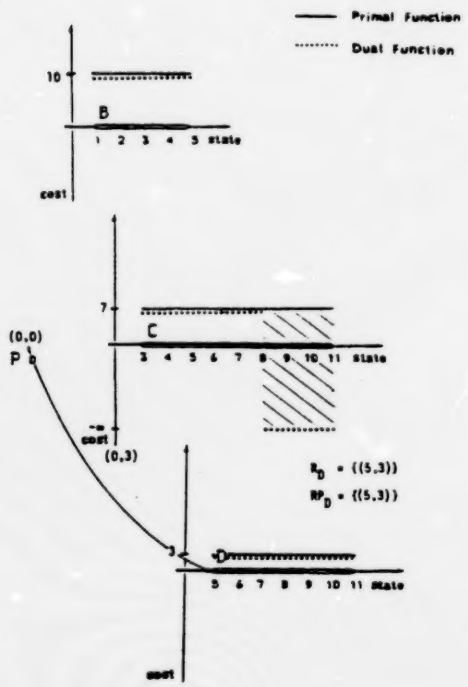

Figure 6
Pulling of label $(5,-\infty)$
at node D

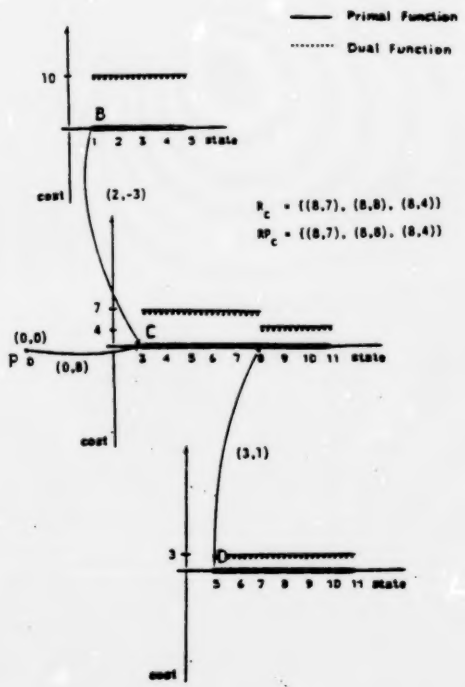

Figure 7
Pulling of label $(8,-\infty)$
at node C

7. CHOICE OF THE ZONE TO BE EXPLORED

Earlier, an example of a zone to be explored was given. However, no precise definition has been provided. As the zone to be explored begins with the appearance of a difference between the values of the primal and dual functions, and is bounded by these functions, we only need to define the state of the end of the zone to define completely the zone to be explored.

Suppose $TR_j = \text{minlex}\,((b_j^{\cdot}, \{T_j^{\cdot m} \mid (T_j^{\cdot m}, C_j^m) \in P_j \text{ and } T_j^{\cdot m} \not\geq T_j^{\cdot k}\})$ where TR_j is the first state in the zone of uncertainty. TR_j corresponds to the arrival state of the first feasible path that is lexicographically posterior to the appearance of the uncertainty. It defines the largest zone to be explored corresponding to the complete zone of uncertainty. The zone defined in this way may be very large compared with the reduction of the zone of uncertainty guaranteed by theorem 1. The construction of the upper bound is mainly efficient at the beginning of the zone of uncertainty. The use of a larger zone of exploration than the guaranteed reduction in uncertainty can also result in a series of replacements before uncertainty is completely eliminated. For example, if the width of the zone is N and the guaranteed reduction is 1, after the first iteration, the dimension of the zone will be N-1 units, and after the second iteration, the dimension of the zone would still be N-2 units. In the worst case, the last part of the interval may be treated N times.

To avoid repeated replacements, it is better to use a fixed length based on the guaranteed reduction in uncertainty, or a variable length based on this value. For example :

$$TSUP_1 = \text{minlex}\{T_j^{\cdot k} + \alpha d_{\cdot j}^{\cdot}, TR_j\}$$

where $\alpha > 1$ and $(d_{\cdot j}^{\cdot}, c_{\cdot j}) = \underset{(i,j)\in A}{\text{minlex}} \{(d_{ij}^{\cdot}, c_{ij})\}$.

This criterion for the definition of the zone of uncertainty ensures that no part of the zone of uncertainty will be examined more than α times, producing a more efficient algorithm.

It would seem reasonable to use $\alpha = 1$ to minimize the complexity of the algorithm. However, a study of the effect of α an execution time [DESR85b] for the case with one additional constraint showed that a value of α between 2 and 3 reduced execution time by around 25% in comparison with α=1. This reduction in execution time is due to the reduction in the minimum number of label replacements which must be carried out. However, if α becomes too large, each replacement becomes very inefficient. We therefore choose α=2 in our implementation.

8. IMPLEMENTATION AND COMPLEXITY

The complexity of the algorithm proposed here will be compared in terms of time and memory requirements with the generalized permanent marking algorithm presented by Desrochers and Soumis [DESR85a] for a single constraint. The following notation will be used throughout this comparison :

L : the number of additional constraints,

$n = |N|$: the number of nodes in the graph G,

$v = \max_{i \in N} \{ \prod_{\ell=1}^{L} (b_i^\ell - a_i^\ell + 1) \}$: the maximum number of states at a node,

$V = \sum_{i=1}^{L} \prod_{\ell=1}^{L} (b_i^\ell - a_i^\ell + 1)$: the total number of states.

These reaching and pulling algorithms may be implemented using two basic approaches which differ in the amount of central memory required to represent the states. The first approach requires the storage in central memory of the labels associated with all possible states in a table of dimension V providing direct access to the labels. The second approach stores only efficient labels in central memory using a self-adjusting

binary tree (or SPLAY) data structure developed by Sleator and Tarjan [SLEA83]. This provides access to an element and its successor. Sleator and Tarjan show that the complexity of any sequence of m operations on a tree with v labels (with $m > v$) is of order $O(m \log v)$. This result allows access to a label in logarithmic time on average. According to the results obtained previously, only 5% to 15% of labels are efficient, so the use of this tree generates considerable memory savings.

The reaching algorithm for the special case with only one additional constraint can be easily generalized and the analysis of complexity carried out by Desrochers and Soumis [DESR85a] can be easily adapted to the case with L constraints. The complexity for the case with one constraint is of the order of $O(nV)$ when a table is used, and of the order of $O(V^2)$ when a pointered list is used. The only difference is that each calculation or comparison of labels requires $O(L)$ operations instead of $O(1)$ in the case studied here. The complexity of the reaching algorithm is therefore of the order of $O(nVL)$ when a table is used, while the complexity with a self-adjusting tree or a pointered list is of the order of $O(V^2L)$.

The complexity of the pulling algorithm depends on the implementation approach adopted. However, the complexity of the first four steps of the algorithm is independent of the data structure chosen. In all, these four steps require $O(|A|L+nL+VL.\log n)$ operations.

Initialization during step 1 prepares the sets P_i et Q_i. The construction of a heap including the lexicographically minimal elements of each of the sets Q_i requires $O(Ln\log n)$ operations. At step 1, it is also necessary to determine, by an examination of all the arcs, the lexicographically minimal arc at each node. This requires $O(|A|L)$ operations. In the worst case therefore, the complexity of step 1 is $O(L.(|A|+n+n\log n))$ operations.

The use of a heap allows rapid access to the lexicographically minimum element of Q_i and thus simplifies steps 2 and 3. In the simplest case, the choice of the label to be replaced merely involves updating the heap and accessing its top.

If it is possible to delay the replacement of a label, the labels whose replacement is postponed are added (in lexicographic order) to a queue of elements of the actual bucket.

In this case, step 2 consists of defining the limits of the bucket by finding the smallest label to be replaced through a comparison between the label at the top of the heap, and the first label in the queue. Step 3 consists of updating the heap, and choosing the next label to be replaced, i.e. the top of the heap, if this belongs to the actual bucket, or if not, an element of the queue.

In both cases, these steps are executed V times. Each execution requires O(L.logn) operations to update the heap and to choose the label to be treated. Overall, steps 2 and 3 require O(V.L.logn) operations in the worst case.

The fourth step may be executed at most V times and the calculation of the boundaries of the zone to be explored requires O(L) operations. Overall, step 4 requires O(VL) operations in the worst case.

Step five may be implemented in three different ways depending on the data structure used, and the algorithm used to test dominance.

The use of a table of dimension V to store all possible labels provides direct access to each label and easy dominance testing between two labels at a given node. The construction of lower and upper bounds in the zone of uncertainty requires the calculation of the cost of each of the paths coming from the m predecessors whose state lies in the zone of

uncertainty. This calculation requires access in constant time to the labels of the predecessors. Dominance testing in a table reduces to a verification of whether the cost of the path is less than the present cost. As each label is treated at most once if α=1, step 5 requires O(V.n.L) operations overall for the replacement of all the labels eliminated at the n nodes i.e. the same complexity as the corresponding approach used for the reaching algorithm.

The use of a self-adjusting binary tree [SLEA83] to store efficient labels permits access to labels in logarithmic time on average. The use of a v-dimensional table to represent the labels associated with states in the zone of uncertainty simplifies dominance testing, and provides some of the advantages of a table while retaining the space savings of the tree structure. The construction of lower and upper bounds in the zone of uncertainty requires the calculation of each path coming from a predecessor of the node, whose associated label is within the zone of uncertainty. The logarithmic access to the labels of predecessors requires in average O(logv) operations. This average can be used as the number of accesses to labels is O(v.number of successors of a node) which is greater than O(v). As the dominance test is carried out in the local table, it is sufficient to check whether the cost of the path is less than the present cost of the state associated with the path. During the construction of the new sets Q_j et P_j, dominance must be verified between the labels in the table and the self-adjusting binary tree must be reconstructed. Overall, given that there is a total of V labels and that each label is treated a maximum of α times, step 5 requires $O(VL\alpha(n\log v+v))$ operations for the replacement of the labels Q_j at the n nodes, where $\alpha > 1$.

The overall complexity of the second approach to implementation is of the order of O(VL.(nlogv+v)) for pulling, and of the order of $O(V^2L)$ for reaching. As O(nlogv+v) < O(V), the complexity of this approach to implementation is less for pulling than for reaching.

The third possible approach also uses a self-adjusting binary tree to store the efficient labels. However, the dominance test is not carried out using a table, but rather using an algorithm for calculating the minimum of a set of vectors (i.e. multi-dimensional labels) [KUNG75]. The complexity of this algorithm in the worst case is of the order of $O(\alpha v(\log v)^{L-2})$ at each node. Overall, for the n nodes, dominance testing requires $O(\alpha V(\log v)^{L-2})$ operations, and access to predecessors requires $O(VL\alpha \log v)$ operations. In total, step 5 requires $O(V\alpha(nL\log v+(\log v)^{L-2}))$ operations ($\alpha > 1$) using this implementation approach.

The overall complexity of the third implementation approach is of the order of $O(V\alpha.(nL\log v+(\log v)^{L-2}))$ for pulling and $O(V^2L)$ for reaching. (N.B. The comparison is carried out with approach 2 for the reaching algorithm as approach 3 offers no advantages here). Once again, the complexity of the pulling algorithm is less than that of the reaching approach.

9. NUMERICAL RESULTS

The pulling algorithm presented in Section 5 was implemented in FORTRAN77 using two of the three data structures suggested in Section 8 with $\alpha=2$. PULL1 uses binary self-adjusting trees to store efficient labels and a table for dominance testing between labels. PULL2 also uses binary self-adjusting trees to store efficient labels, but uses a specialized dominance testing algorithm. The two versions use all the same subroutines except those concerned with dominance testing. They were compiled by the FTN5 compiler with the option OPT=2 on the CDC CYBER 173 computer of the Université de Montréal. The version PULL1 cannot be used if the number of states at each node (i.e. the size of the table used for dominance testing) is too large. The comparison of the two methods must therefore restricted to problems where size is relatively low. There is no operating version of the reaching algorithm for the SPPRC. However, the comparison carried out

in [DESR85b] between pulling and reaching for the case with one additional constraint shows that pulling is slightly faster than reaching.

The comparison between the two versions will be carried out on some typical problems arising in the construction of workdays for urban bus drivers. The aim is to produce a workday beginning between times A and B which respects 3 to 5 additional constraints. A workday is made up of one or more tasks separated by breaks and must respect the following constraints :

1. the number of tasks is limited;
2. the total duration of the workday (including tasks and breaks) is limited;
3. the number of working hours (i.e. the duration of tasks executed) is limited.

For the problems with five additional constraints, two other resources are limited.

For each of these constraints, the amount of a resource used to reach a task must lie within a feasible window associated with the task and the resource. In our type of problem, in two cases (number of tasks executed, and number of hours worked), the window is of the form [0..bsup]. In the three other cases (total duration and resources 4 and 5), the window is adapted to each task. The window on the total duration for a task beginning at time d is [d-b..d-a].

The order of elements within the vectors $(d^1_{ij},\ldots,d^L_{ij})$ is important for determining the lexicographic order, and this order can affect the efficiency of the algorithm. We therefore choose as the first element of the vector, the element ℓ such taht : $\Theta^{\ell} = \left\{ \frac{b^{\ell}_j - a^{\ell}_j + 1}{\min\limits_{(i,j)\in A} d^{\ell}_{ij}} \right\}$ is minimized. In

order to minimize the maximum number of pullings at each node, θ^{ℓ} represents the maximum number of generalized buckets if d^{ℓ}_{ij} is the first element of the vector.

We generated two series of 5 problems for each of four problem sizes : N = 100, 250, 500, 1000. The characteristics of these problems (i.e. $[a^{\ell}_i, b^{\ell}_i]$, $i \in N$, $\ell=1,\ldots,5$ and c_{ij}, d^{ℓ}_{ij}, $(i,j) \in A$, $\ell=1,\ldots,5$) were generated according to uniform distributions so as to simulate workday construction problems. The first series represents problems in which the data have been rounded up to reduce the amount of memory v required at each node, while in the second series, the data are more precise. This enables us also to study the effect of v on the behaviour of the two versions. We also doubled the series by extracting problems with L=3.

Tables 1 and 2 provide descriptions of the problems generated. $|N|$ is the number of nodes for each problem. $|A|$ is the mean number of feasible arcs in the network. It should be noted that the number of feasible arcs is greater in the problems with three constraints than those with five constraints. This is due to the additional feasibility checks (6). In the tables, v is the number of states at a node, while V is the total number of states. V is the maximum possible number of labels. $|E|$ is the number of labels associated with efficient solutions in the final solution. It is not unusual for the number of labels present in the solution during the execution of the algorithm to be greater than $|E|$.

The statistics on execution presented in Tables 1 and 2 are the number of pullings and the execution time. The computer used did not allow storage of all the nodes in central memory so the arcs were divided into pages and only one page was stored in central memory. When the execution of the algorithm required arcs from another page, this new page was brought into central memory. This process resembles virtual memory use on other computers. Memory management time was measured separately from algorithm

TABLE 1

DATA DESCRIPTION					PULL1			PULL2		
|N|	|A|	V	|E|	#PULL	Time ES (sec.)	Time UC (sec.)		Time ES (sec.)	Time UC (sec.)	
					Mean	Max	Mean	Mean	Max	Mean
100	1566	9224	164	101	0,000	1,551	1,214	0,000	1,011	0,890
250	9096	23086	526	251	0,023	6,029	4,612	0,024	5,014	4,294
500	36241	46210	1290	501	0,348	21,717	16,055	0,349	23,071	17,411
1000	144082	92336	2998	1001	1,830	86,674	59,645	1,846	111,348	69,747

Execution statistics for the two versions for problems with three contraints and v varying between 1 and 220 (mean =92).

|N|	|A|	V	|E|	#PULL	Time ES (sec.)	Time UC (sec.)		Time ES (sec.)	Time UC (sec.)	
					Mean	Max	Mean	Mean	Max	Mean
100	1746	42724	348	101	0,000	4,953	3,235	0,000	1,253	1,097
250	10282	107166	1657	251	0,070	17,890	10,509	0,071	13,732	6,973
500	40836	214416	4578	501	0,532	60,657	34,586	0,438	77,735	31,816
1000	insufficient memory									

Execution statistics for the two versions for problems with 3 constraints and v varying between 1 and 1000 (mean = 427)

TABLE 2

DATA DESCRIPTION					PULL1			PULL2		
|N|	|A|	V	|E|	#PULL	Time ES (sec.)	Time UC (sec.)		Time ES (sec.)	Time UC (sec.)	
					Mean	Max	Mean	Mean	Max	Mean
100	582	133833	83	101	0,000	18,756	10,522	0,000	0,602	0,456
250	3364	325658	401	251	0,000	47,211	27,211	0,000	2,479	2,062
500	12030	646179	1154	501	0,095	97,826	55,956	0,096	9,677	7,050
1000	46889	1295252	2913	1001	0,510	205,787	121,025	0,543	39,681	26,371

Execution statistics for the two versions for problems with five constraints and v varying between 1 and 9200 (mean = 1298)

|N|	|A|	V	|E|	#PULL	Time ES (sec.)	Time UC (sec.)		Time ES (sec.)	Time UC (sec.)	
					Mean	Max	Mean	Max	Max	Mean
100	641	602812	123	101	insufficient memory			0,000	0,686	0,517
250	3742	1511336	729	251				0,000	3,744	2,592
500	13423	2998202	2513	501				0,121	22,712	10,959
1000	52978	6009189	7641	1001				0,636	104,709	46,806

Execution statistics for the two versions for problems with five constraints and v varying between 1 and 46000 (mean = 6012)

execution time. However, as memory management was carried out by routines common to both versions of the algorithm, no significant difference can be observed in memory management time for the two approaches.

Execution statistics for the 16 series of problems are shown in Tables 1 and 2. The first observation is that PULL2 solved 15 of the 16 series of problems, while PULL1 solved only 11 out of 16 series because of memory difficulties. The series which could not be solved by either method required much greater memory than the others. The two versions seem to be complementary. PULL1 is slightly better than PULL2 on problems with L=3 and where v is small. But this advantage disappears as v increases. When L=5, PULL2 clearly dominates, requiring only 5% to 20% of the computation time required by PULL1 and using less central memory.

The difference in the execution times of PULL1 and PULL2 can be explained in terms of the variables present in the analysis of worst case complexity. For PULL1, complexity is of the order of $O(VL(n\log v+v))$, while for PULL2, it is of the order of $O(V(nL\log v+(\log v)^{L-2}))$. The $O(VLn\log v)$ part is the same for the two versions. Only the $O(VLv)$ part for PULL1 and the $O(V(\log v)^{L-2})$ part for PULL2 are different. Dominance testing requires at least $O(v)$ operations for PULL1, while for PULL2 it requires $O(k\log^2 k)$ where k is the number of vectors obtained ($k<v$). In the case where v is large and k is small, $O(v)$ is often greater than $O(k(\log k)^{L-2})$, which explains why PULL2 is faster than PULL1 in the case where L=5.

The performance of PULL2 is surprising in one sense : its execution time diminishes as the number of constraints increases from 3 to 5 and the number of states V increases significantly). This reduction can be explained by the reduction in the number of feasible arcs in the problems producing a reduciton in the number of states examined. The number of operations in PULL2 depends on the number of labels examined rather than on the number of states V. PULL2 can solve highly constrained problems in a reasonable time.

10. CONCLUSION

A new algorithm for the constrained shortest path problem was developed to solve routing and scheduling sub-problems. It can also be used to solve knapsack type problems with one or more resources using windows $[0,b^{\ell}]$ at all the nodes. In addition, we have already seen that this algorithm can be used for reoptimization after cost increases.

The pulling approach, which finds the solution at a node by consulting the solution at several predecessor nodes, seems more suited to parallel processing computer implementation than the reaching approach which modifies the solution at several successor nodes when the slution at a given node is consulted.

The numerical results lead to one surprising conclusion : for PULL2, computation time seems to be influenced more by the number of solutions to the problem than by the number of states. In fact, the addition of two additional constraints to problems with L=3 increases greatly the number of states V but reduces the number of arcs and the number of feasible solutions to the problem. We therefore observe a reduction in execution time between problems with L=3 and L=5 when the number of constraints increases. This, and the possibility of implementation on a multi-processor computer opens the way to the solutio of large, very complex and highly constrained problems in a reasonable time.

REFERENCES

[ANEJ 83] ANEJA, Y.P., AGGARWAL, V. et NAIR, K.P.K., Shortest chain subject to side constraints. Networks 13 (1983) 295-302

[BEAS 84] BEASLEY, J.E. and CHRISTOFIDES, N., An algorithm for the resource contrained shortest path problem, Working paper, Department of Management Science, Imperial College, London, 24 pages (1984).

[DESR 85a] DESROCHERS, M. et SOUMIS, F., A generalized permanent labelling algorithm for the shortest path problem. Publication #394F, Centre de recherche sur les transports, Université de Montréal, 28 pages (1985).

[DESR85b] DESROCHERS, M. et Soumis, F., A reoptimization algorithm for the shortest path problem with time windows. Publication #397F, Centre de recherche sur les transports, Université de Montréal, 25 pages (1985).

[DESR 83] DESROSIERS, J., PELLETIER, P. et SOUMIS, F., Plus court chemin avec contraintes d'horaires. R.A.I.R.O. Recherche Opérationnelle 17 (1983) 357-377.

[DESR 84] DESROSIERS, J., SOUMIS, F. et DESROCHERS, M., Routing with time windows by column generation. Networks 14 (1984) 545-565.

[GARE 79] GAREY, M.R. and JOHNSON, D.S., Computer and intractability : a guide to the theory of NP-completeness. Freeman (1979) San Francisco.

[HABE 81] HABENICHT, W., Efficient routes in vector-values graphs. Proceedings of the Seventh Conference on Graphtheoretic Concepts in Computer Science, J.R. Mühlbacher, ed., Carl Hanser Verlag, Munich (1981) 349-355.

[HANS 80] HANSEN, P., Bicriterion path problems, dans G. Fandel et T. Gal, éds., Multiple Criteria Decision Making : Theory and Applications, Lecture Notes in Economics and Mathematical Systems 177, Springer Verlag, Heidelberg (1980) 109-127.

[JAFF 84] JAFFE, J.M., Algorithms for finding paths with multiple constraints. Networks 14 (1984) 95-116.

[JOKS 66] JOHSCH, H.C., The shortest route problem with constraints. Journal of Mathematical Analysis and Applications 14 (1966) 191-197.

[KUNG 75] KUNG, H.T., LUCCIO, F., PREPARATA, F.P., On finding the maxima of a set of vectors, JACM 22, 5 (1975) 469-476.

[MART 84] MARTINS, E.Q.V., On a multicriteria shortest path problem. European Journal of Operational Research 16 (1984) 236-245.

[MINO 75] MINOUX, M., Plus court chemin avec contraintes : algorithmes et applications. Annales des Télécommunications 30, 12 (1975) 1-12.

[SLEA 83] SLEATOR, D.D., TARJAN, R.E., Self-adjusting binary trees, Proc. Fifteenth Annual ACM Symposium on the Theory of Computing (1983) 235-245.

[VINC 74] VINCKE, P., Problèmes multicritères. Cahiers du Centre d'Études en Recherche Opérationnelle 16 (1974) 425-439.

[WARB 86] WARBURTON, A., Approximation of Pareto optima in multiple objective shortest path problems, Working paper, University of Ottawa (1986) 36 pages.